RULEBASED INFERENCING OVER THE WORLD-WIDE-WEB

The aide Developers Manual

Russell Allen, Damien Piper, Graham Greenleaf,

Andrew Mowbray, and Philip Chung

March 2001

Table of Contents

1. Introduction

This tutorial aims to provide a practical introduction to rule-based inferencing systems, and to issues in inferencing with legal sources. It covers the development of a simple backward chaining rulebase using aide, rule structures, aide's grammatical structure, collaborative development using aide and other features of the current version of aide.

Anyone who wishes to use these resources for teaching or other non-commercial purposes should contact AustLII at feedback@austlii.edu.au.

1.1. The aide index page

Most of the resources needed for the activities in this manual can be located from the aide wbsite on AustLII: http://aide.austlii.edu.au

2. Demonstrations using wysh

To demonstrate legal infrerencing, we will use some applications developed using the AustLII's old software *ysh* (pronounced 'why - shell') and *wysh* (pronounced 'wish' and standing for 'web-ysh'); Ysh is the underlying inferencing engine, and wysh is the web interface to ysh. For convenience, we will refer to both simply as 'wysh' for most purposes.

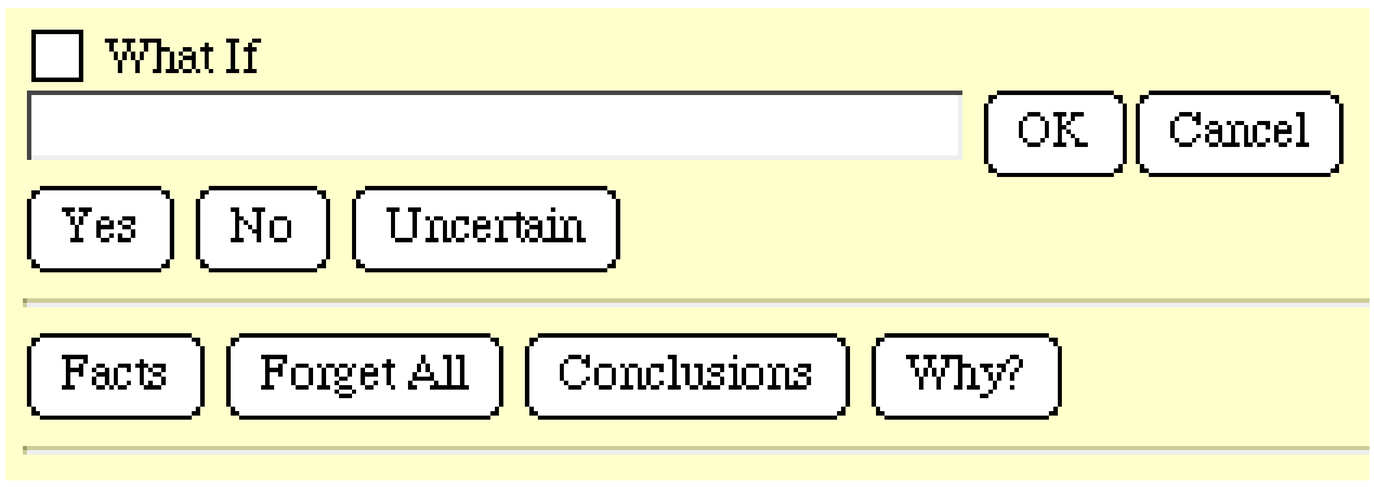
Both systems are the result of previous AustLII inferencing projects. Even though the aide project differs in many ways to both ysh and wysh, it could still be useful to read the literature for both previous systems. *The Wysh User Manual*, AustLII, 1997 and *The Wysh Developer's Manual*, AustLII, 1997, are two important pieces.

2.1. Demonstrations of inferencing using ysh / wysh

The purpose of this exercise is to familiarise you with the basic concepts and applications involved in inferencing. Examples on copyright and privacy law are used to illustrate some rule-based applications that can be developed.

2.2. The wysh user interface - elements to test

When doing the following examples, please test the use of all the system functions, including the buttons for'Facts' (what have you told the system), 'Conclusions' (what conclusions has the system derived), 'Why' (why is the current question being asked), 'What if' (what conclusions will be derived from a hypothetical fact) and 'Uncertain' (if a fact is not essential, inferencing will continue). You may also wish to refer to the *ysh/wysh User Manual* , but there is little need.



Things to note about the interface:

- To choose a goal to be evaluated, type the number of that goal in the window;

- If ‘Yes’, ‘No’ or ‘Uncertain’ buttons do not appear, you can type ‘y’, ‘n’ or ‘u’ in the window.
- To forget a single fact, first display the facts (using the ‘Facts’ button), the enter ‘forget n’ (where n is the number of the fact to be forgotten) in the window (eg ‘forget 11’ will cause fact 11 to be forgotten and all conclusions re-derived;
- Do not use the Netscape ‘back’ button in order to change facts - it won’t work and may give bizarre results

2.3. Privacy and copyright examples

The ‘Copyright Consultations’ <http://aide.austlii.edu.au/copyright/> and ‘Privacy Consultations’ <http://aide.austlii.edu.au/wysh/privacy.html> on the [wysh] index page should be used to answer the following questions. If you need more facts than are provided, make them up.

Lost in space

- Harold Henning, a famous Australian author, wrote a short story ‘Lost in Space’ on Xmas Day 1940 when he was young. It has sat in his desk draw ever since. His ‘unofficial biographer’, Fiona Fail, has obtained a copy and intends to publish it as an appendix to her book about him. Does Harold still hold copyright, and if so until when?
- What if Harold died in 1950?
- What if a radio adaptation of the short story had been broadcast once in 1945? Does it matter in this case if Harold is still alive?

Freelancer

- Jana, a freelance journalist, agrees verbally to write a weekly column ‘Around the Courts’, for a monthly magazine ‘TV Times’. She has no discussions about copyright issues with Rupert, the owner.

Roll your own

Before you leave the copyright problems, make up one of your own and test it. Do the same after you finish the privacy problems below.

Privacy problems

- Is it legal for a credit bureau like the Credit Reference Association of Australia to hold on file details of a person’s racial origin?
- If the Department of Social Security holds your address in its files, can it disclose that address to the NSW Department of Housing, which wants to serve a summons on you? (Hint: IPP 11 is the most relevant Information Privacy Principle)

3. Backward & forward chaining rules

Some simple symbolic examples are used to make it easier to see the logic behind backward and forward chaining without the distraction of real facts.

3.1. A set of rules ...

Assume that you have 5 rules which allow you to reach conclusions about whether 8 propositions (represented as A to F) are true or not:

Rule	***Conditions***	***Conclusion***
Rule 1	E and F	A
Rule 2	A and (B or C)	D
Rule 3	E or G	C
Rule 4	H and G and C and not F	A
Rule 5	F and not E	not D

For example, Rule 1 means 'If condtion E and condition F are both true, then conclusion A is true.'

3.2. ... and a problem

f you were asked whether D is true or not, given that you know that E, H and G are true, but F and B are false, how would you go about working out an answer. **Write down each step in your reasoning.**

3.3. Forward and backward chaining - testing ysh

There are 3 wysh files that you can use, entitled 'Goal Tests' <http://www2.austlii.edu.au/~graham/wysh/goaltest.html>:

- *goaltest* - forward and backward chaining
- *goalback* - backward chaining only
- *goalfor* - forward chaining only

You can now use these files to test how YSH's inferencing mechanism works. For example, assume the following:

- the objective (goal) is D;
- E, H and G are correct (true);
- F and B are not correct (false).

First, try to work out on paper what YSH should do, using goaltest (forward and backward chaining) to start with, then watch to see if it does it. Then try out objective and fact combinations of your own, and try the versions which just do forward or backward chaining and observer the differences in behaviour.

Note the order in which rules and attributes are evaluated in YSH's backward-chaining mode:

- rules are evaluated top-down from the start of the rule-base; and
- attributes are evaluated left to right within a rule

Backwards chaining is the default approach for *aide*.

4. How is *aide* different to wysh?

The main similarity between wysh and *aide* is that they are inferencing tools that are used primarily to develop legal rulebases. They also provide facilities for collaborative rulebase development. Beyond these conceptual similarities, there are a number of major differences between the two systems.

4.1. Capability

Wysh is basically an expert system compiler. It allows the user to input hyperlinks into webpages that contain embedded rulebases. The rulebases are created using a HTML or text editor. Wysh would then compile all the linked rules together for each session.

aide provides a common editor for rule creation. This was seen to be necessary because rules contain a structured arrangement of elements, and the user needed to distinguish between the elements when creating and modifying the rules. The aide editor window provides a number of useful features necessary for developing a correctly structured rulebase. Such features are described in more detail later.

4.2. Predicate Logic - Reasoning with objects

The biggest and most significant difference between the two systems is their approach to clauses. Wysh takes the clauses between the keywords as a whole and does not attempt to interpret the elements within each separate clause. This approach is known as propositional logic. Propositional logic makes rule creation fairly simple, as it only requires knowledge of the keywords and their relationship. However, it can make consultations very confusing. Propositional logic takes each clause as a whole and treats it as a symbol, for example:

```
“it is raining” = P

“I am asleep” = Q

“the alarm is ringing” = R

“I will sleep in” = Z
```

A propositional logic rule could then be stated as:

```
IF P and Q and R THEN Z.
```

The approach of treating a clause as a whole has some limitations. Questions often are confusing due to the unpredictability of the English language, caused by the flexibility of English expression. Also the subject of the consultation can often be confused, for example a user may forget what document he or she was referring to, say 20 questions ago.

aide overcomes these common problems by attempting to interpret each element within a clause. This approach is called predicate logic. Predicate logic requires a clause to be dissected into individual components, which can be used to infer new sentences. Each individual component can be though of an object. Predicate logic allows multiple instances, or occurrences, of the same object within the one session. For example take the following rule:

```
If an animal can bark then the animal is a dog.
```

Wysh would require this rule to be written as:

```
IF an animal can bark THEN the animal is a dog.
```

Wysh would collect “an animal can bark” into one variable and “the animal is a dog” into another. aide also recognises “If” and “then” as keywords, however it would attempt to collect “an animal” into one variable, “can bark” into another, “the animal” into another, “is” into another, and “a dog” into another. The process of slicing up each clause into elements is called **parsing**. In the above example there are 3 types of elements. They are collected as:

- **Objects** – “an animal” and “the animal” – both are referring to the same object;
- *Verb* – “can bark” and “is”; and
- Attributes – “a dog” – this is a characteristic of the object. [Optional]

The benefit of this approach is that in one session many dogs can be referred to.

The following is the original rule split into the three main elements.

```
If an animal can bark then the animal is a dog.
```

4.3. Syntax

Wysh has a larger syntax than *aide* does. This is due to wysh’s collective approach compared to aide’s object-oriented approach. Later versions of *aide* may include more keywords, because further features may require the use of keywords.

5. First steps: *aide* and its Value

The purpose of this exercise is to familiarise you with the *aide* software. Examples on copyright law are used to illustrate some rule-based applications that can be developed.

5.1. Elements of the aide Package

aide has two standard functions; firstly, aide can be used as a text editor to create and parse a knowledgebase, and secondly, aide can be used as an inferencing engine to run the rules located in the knowledgebase. The following figure displays the main menu, which is the “control-base” of aide where rules can be controlled.

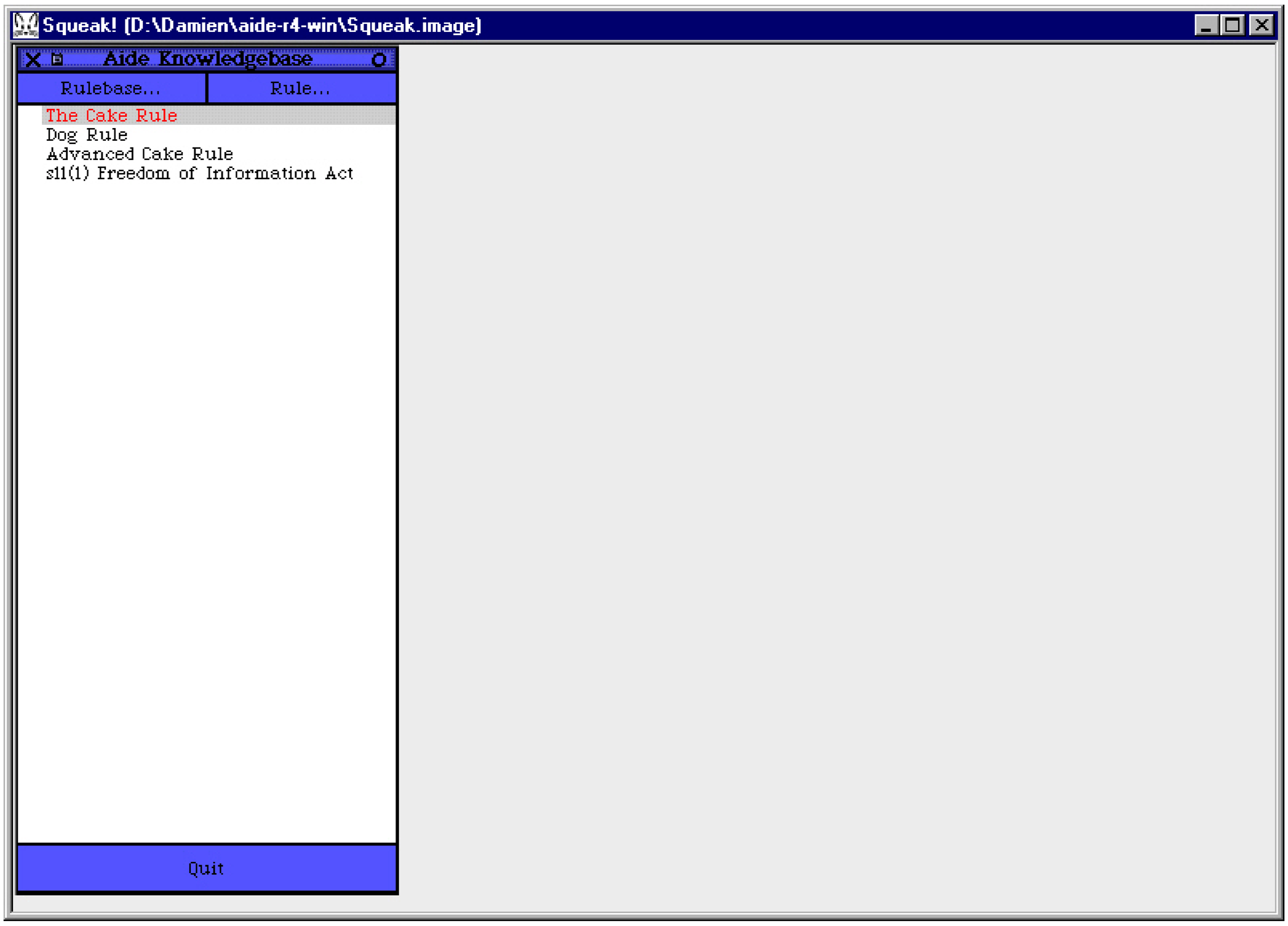


Figure 2.1a – aide Main Menu

The main menu provides a list of the current rules in window. Rulebase and Rule are the two main menu options. The quit button can be used to exit the program and save the current state.

WARNING

NEVER exit *aide* by selecting the standard windows exit button, which is the cross in the top right hand corner of aide. If this is selected the program will quit without saving the changes to the current rulebase.

The main menu bar can be adjusted to any size you feel comfortable with. The bar can also be adjusted by some of the options under the window menu as illustrated below in figure 2.1b.

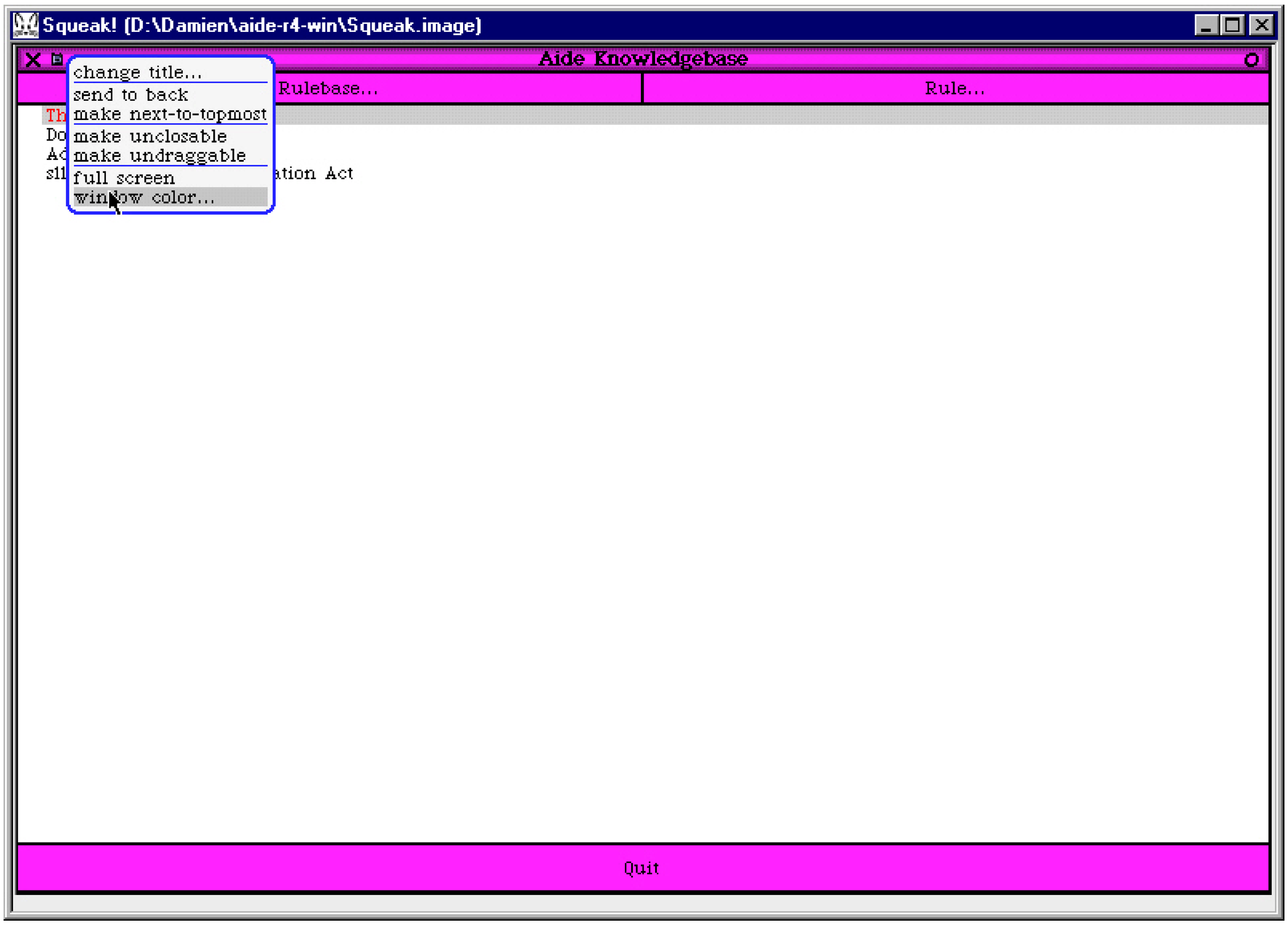


Figure 2.1b – Adjusted Main Menu

Under the rule menu option from the main menu, the user has the option to create a new rule, or if a rule is selected from the list, the selected rule can be edited, deleted or run. Under the rulebase menu option, the user may save or load a rulebase from file, or load a rulebase from the web. These options will be explored a little later.

When you are editing or creating rules, you will be working in the editor window. The following figure displays an editor window, which sits on top of the main menu screen.

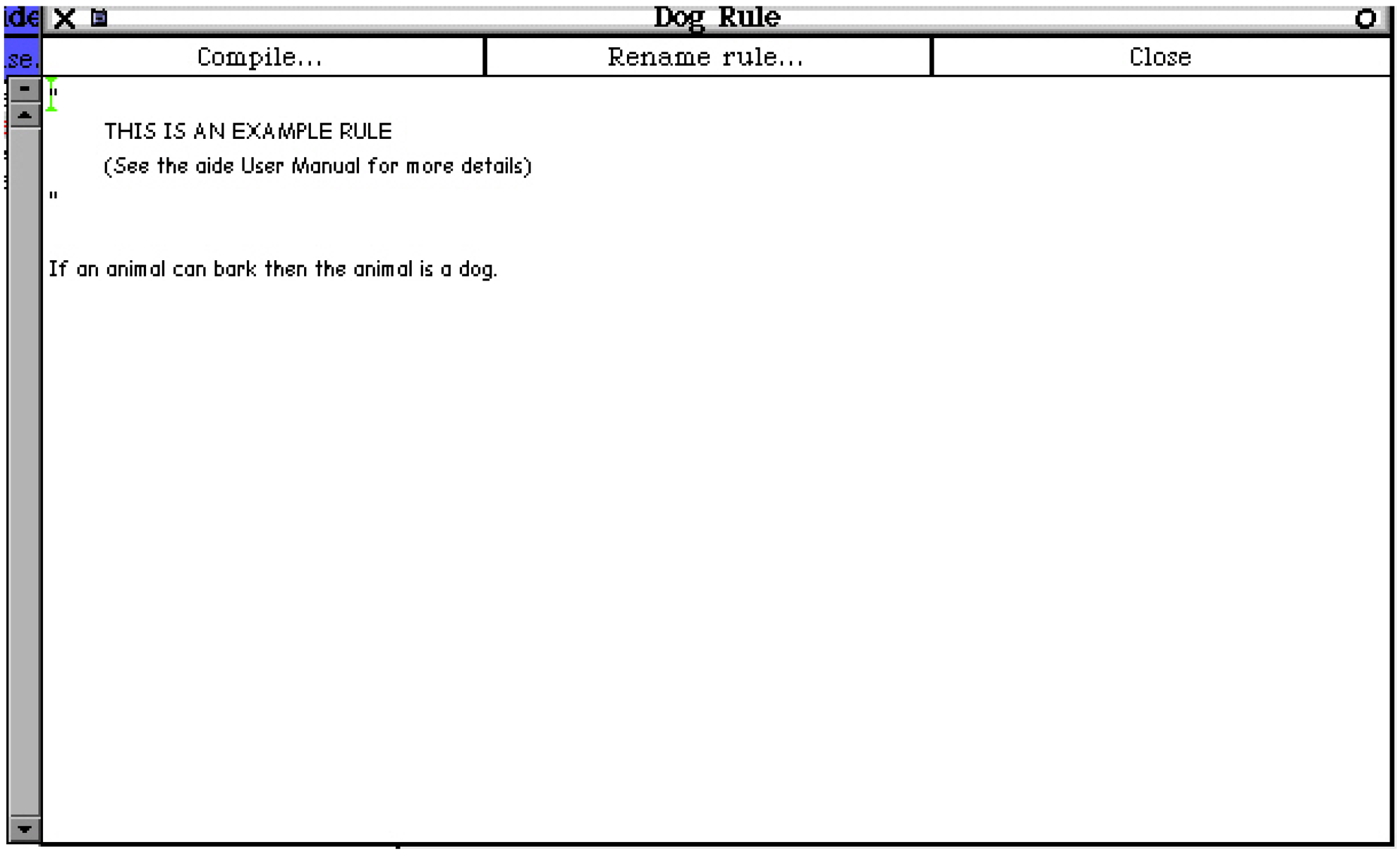


Figure 2.2 – Editor Window

The editor window is a simple text window, which acts like a traditional text-editing program. Text can only be entered when the cursor is hovering over the window. Hence, the cursor indicates what window is currently active. Another different characteristic of this editor window is that the scroll bar is on the left hand side of the window and hidden when the cursor is not hovering over the window.

Once the rules have been entered the compile button needs to be selected, so that the rules are 'parsed', meaning arranged into memory. The close button causes the current editor window to close and the user will be returned to the main menu.

WARNING

It is important to note that the current version of *aide* does not check to see if any new text has been parsed before closing. This means that you will loose any work completed since the rules were last parsed. If the user has chosen to create a new rule, entered the necessary text, then selected close before they selected compile, all the work will be lost.

The rename rule button will allow the rule name to be changed. You do not need to understand any other features before conducting the practical aspects of the following sub-section.

5.2. aide Consultations

Once a rulebase has been created it can be run. Any rule can be run by selecting the rule from the main menu list and selecting rule then run rule. This should cause the consultation window to open. The consultation window is used by aide to collect the facts of the current matter and then display the conclusion. The following figure illustrates a question being asked via the consultation window.

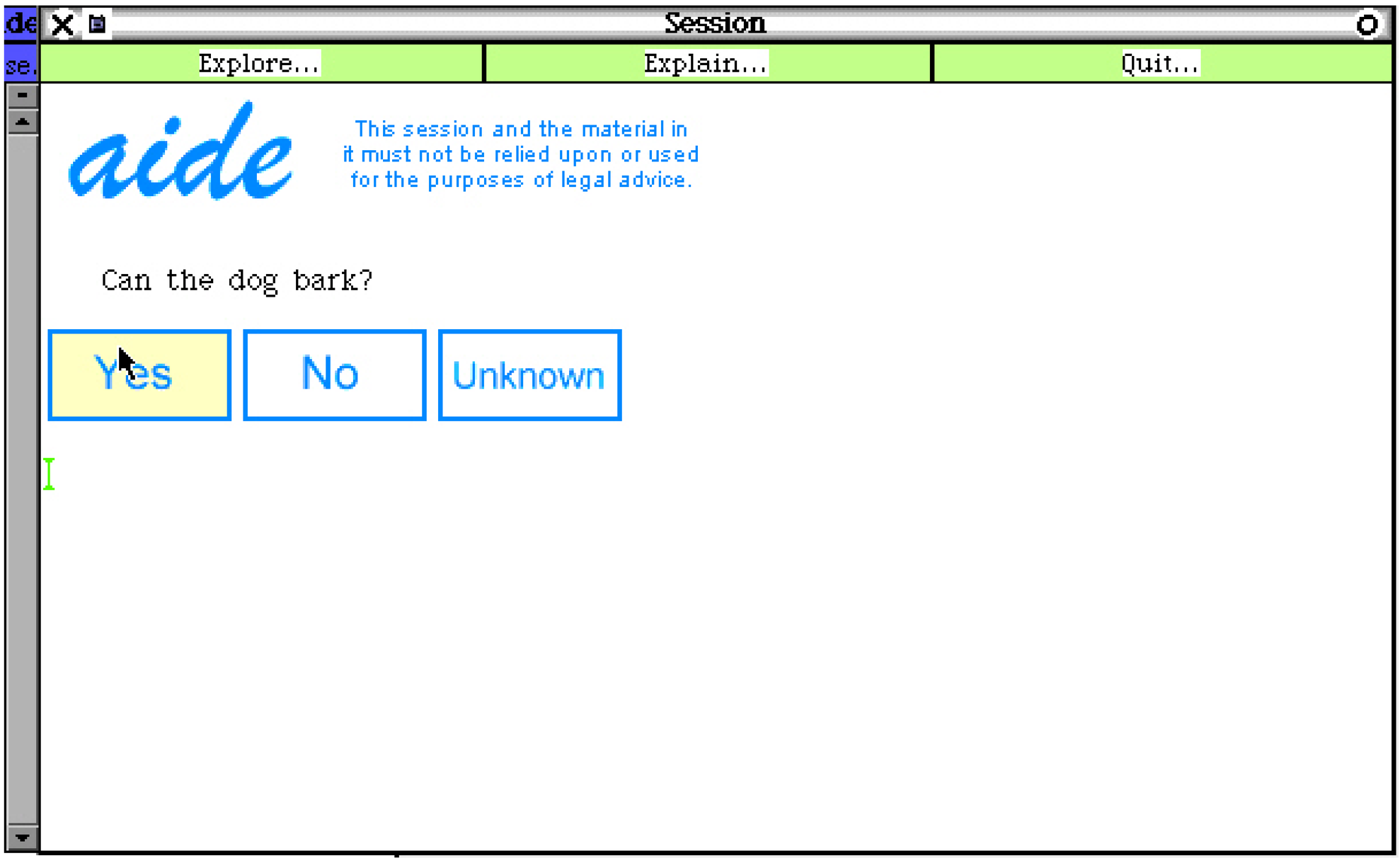


Figure 2.3 – Consultation Window with restricted input

An aide consultation involves answering questions that the inference engine generates from the rules. The questions can often involve a restricted answer, as displayed above, or an unrestricted answer. The figure above shows a question involving a restricted answer, due to the display of three possible responses:

- Yes = True
- No = False
- Unknown = Uncertain

HINT

It is recommended that the user does answer either yes or no, where possible. Although in theory unknown should be working, in practice it does not give correct results in this version of the system.

Often aide needs to ask questions that require an unrestricted answer. This often occurs due to the object-oriented nature of aide. aide often must determine the number of instances/occurrences of each object and the name of each instance. The following figure displays a common form of question with unrestricted input answer.

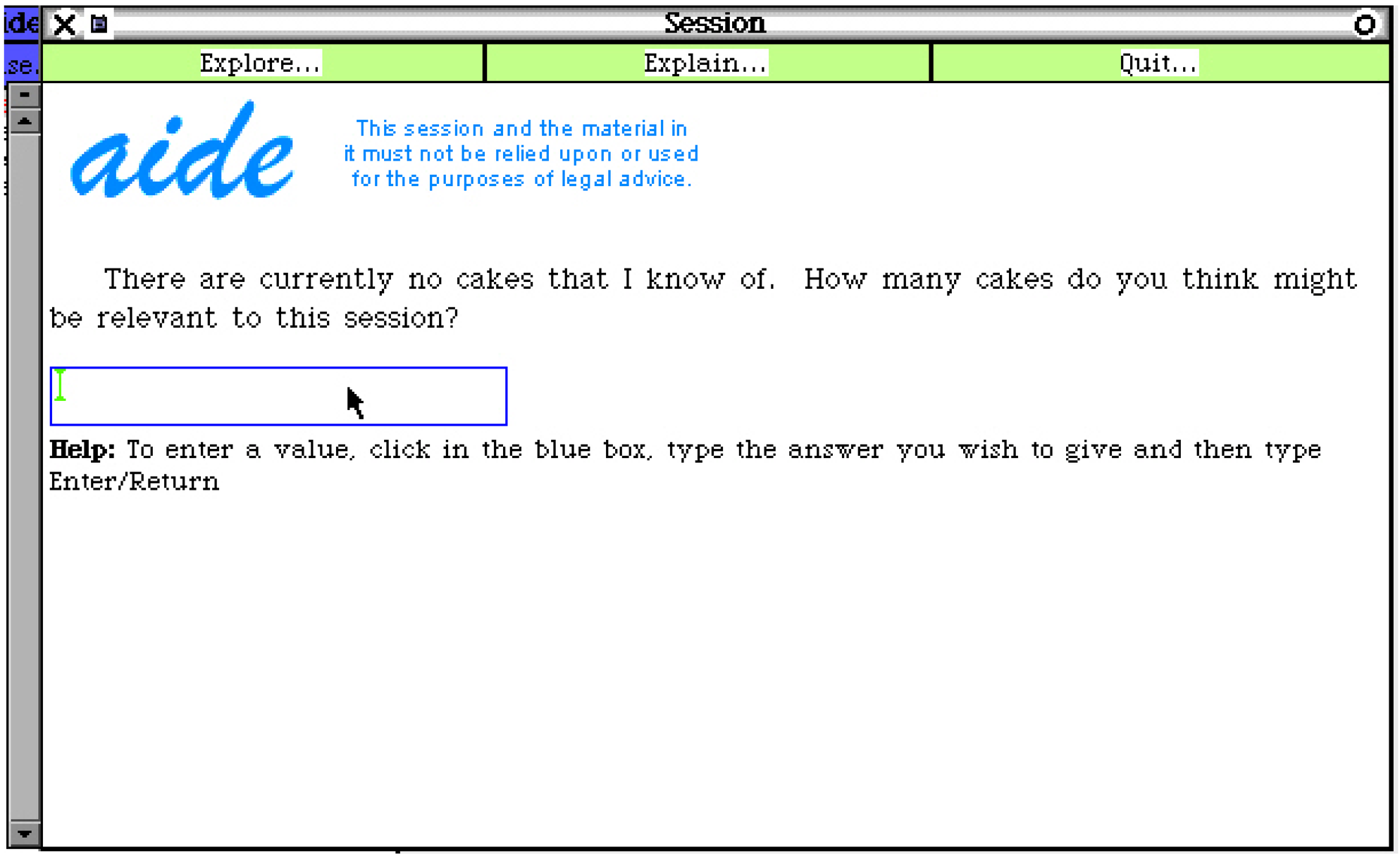


Figure 2.4 – Consultation Window with Unrestricted Input

Here the input is unrestricted in the sense that the user is not restricted to the type of response entered. However, the inference engine will not know how to handle input that does not adhere to the expected input format. In the consultation illustrated by figure 2.4 above, the user could input “Donkeys”, “2 cakes mate” or “1000”, but none of these entries will be understood by the inferencing engine. The user is restricted by the inferencing engine to enter a whole number between 0 and 9, because aide cannot deal with more than 10 instances of the one object (the object being the cake). Common sense must be used when considering what input format could be required.

While completing a consultation, a user may select the explore, explain or quit buttons on the top bar. The explore button allows the user to see the facts and the conclusions that have been found and arrived at in the current system state. The explain button provides a view into the entire state of the inferencing engine, hence it is not restricted to only the facts and conclusions. It should be noted that both of these buttons provide a very low-level interface. Plans have been made to make these tools a little more self-explanatory, so that the novice users are not confused by the mass of detail.

The quit button ends the consultation regardless of the consultation status.

WARNING

Do not select the quit button at any stage during the consultation, except once the result has been displayed. You cannot recover a session that was left by selecting the quit button.

6. Writing Very Simple Rules Using *aide*

The best way to learn how to create rules is by example. The following section describes the process of creating individual rules and linking a group of rules. The purpose of this exercise to explain how to convert a section of an Act into a small rulebase.

We encourage you to read through this section and then practice what you have learnt on a small piece of legislation you are already familiar with. Find the legislation on the web; choose a couple of fairly simple and inter-related sections, preferably sections that create obligations or offences. The easiest way to proceed is to cut and paste the section on which you wish to model into the editor window, and then edit it. We use a mixture of everyday and legislative rules in the following sub-section.

6.1. General Structure

Currently, rules must be sentences in the form:

```
If condition then conclusion.
```

or

```
If condition then conclusion else alternative conclusion.
```

It is important to begin your sentence with a capital letter, and finish it with a full-stop. The current version of aide only allows full-stops to be used to indicate the end of the rule. Hence, you should not number elements like 2.3.1 and so on. Instead you should use hyphens, for example “Rule 2-1-3”.

Here is an example of a rule. This rule is in the example rule set called "Example Rule 1".

```
"

     THIS IS EXAMPLE RULE 1

     (see the aide User Manual for more details)

"


If an animal can bark then the animal is a dog.
```

The text between the double-quotes is treated as a comment, and will not be parsed by the system.

HINT

Note that you cannot nest comments. This means you cannot include comments marks within a wider set of comments marks.

6.2. Clause Elements

In this example, the condition is "an animal can bark" and the conclusion is "the animal is a dog". Both of these are example of a 'clause'. A clause is a grammatical part of the English language. Clauses in aide are more restricted then clauses in general English grammar.

In general, a clause is made up of a subject, a verb, an optional object, and one or more optional phrases. The subject and object are both 'noun phrases'. Here are some valid noun phrases:

```
the dog

a document

the official document

an official document of the Minister

the public

the copy of the cinematograph film

a record embodying a sound recording
```

Here are some invalid noun phrases:

```
copies of the document

the document's author
```

Most verb phrases will parse correctly, even complex phrases such as "will have been published".

HINT

In aide, you must state everything explicitly. This means that you cannot use pronouns such as "it", "he", "they" etc.

6.3. Logic

Rules may have logic constructors in their conditions. A logic constructor may be either 'and' or 'or'. An example of this sort of rule is:

```
"
     THIS IS EXAMPLE RULE 2
     (see the aide User Manual for more details)
"

If the cake is chocolate and the cake is free then the cake
will quickly disappear.
```

In this example, the condition "the cake is chocolate and the cake is free" is made up of two clauses connected by 'and':

```
the cake is chocolate
```

and

```
the cake is free
```

You can chain as many clauses together in this way as you want, but you can only use one type of connector. For example, this is not a valid condition because it uses both the 'and' connector and the 'or' connector:

```
If the cake is chocolate and the cake is free or the cake is
very cheap then the cake will quickly disappear.
```

6.4. Advanced Logic

How then do we express the logic of the rule we considered above? The answer is that we use an 'option list', which is designed to be as similar to legislation as possible. Here is the rule in valid aide syntax:

```
"
     THIS IS EXAMPLE RULE 3
     (see the aide User Manual for more details)
"

If the cake is chocolate and:
(1)  the cake is free; or
(2)  the cake is cheap;
then the cake will quickly disappear.
```

The key points of this are:

7 **each option must begin with an identifier surrounded by brackets '(1)'**

8 **there must be a single tab after the bracket**

9 **all options must finish with a semi-colon ';'**

10 **the second last option must indicate the type of option list (either 'and' or 'or') by placing it after the semi-colon.**

WARNING

These requirements are very strict! If you don't get this right you will get all sorts of strange error messages, and your rule will not compile. This early version of the software is not very tolerant of variations in syntax, a problem we will be trying to fix in later (more stable) versions.

6.5. aide Syntax – FOI Act

s11 of the Commonwealth Freedom of Information Act is a perennial favorite, having been one of the first sections to be translated into ysh syntax back in the mid-1980s. Here is the new aide version:

```
"

FREEDOM OF INFORMATION ACT 1982 SECT 11

Part III—Access to documents 11 Right of access

"


(1)  If there is a person and:

    (a)  the document is a document of an agency and the
document is not an exempt document; or

    (b)  the document is an official document of a Minister
and the document is not an exempt document;

    then the person is legally entitled to access the
document.
```

Here is the original section:

```
(1) Subject to this Act, every person has a legally
enforceable right to obtain access in accordance with this
Act to:

    (a)  a document of an agency, other than an exempt
document; or

    (b)  an official document of a Minister, other than an
exempt document.
```

Look at the details of how this section was translated into the new syntax. Note that since we are starting at the sub-section level (one level below that of the previous cake example), there are two tabs in front of the '(a)', but that there is only one tab after the ‘(a)’. Note also that there is one tab before the 'then' to bring it into line with the 'If'.

One possible trap may be that of putting in extra carriage returns. Only put in carriage returns at the end of an option. Here is the above example, with tabs and carriage returns printed:

```
"↲
FREEDOM OF INFORMATION ACT 1982 SECT 11 ↲
Part III—Access to documents 11 Right of access↲
"↲
↲
(1) →If there is a person and:↲
→ → (a) →the document is a document of an agency and the document is not an exempt document; or↲
→ → (b) →the document is an official document of a Minister and the document is not an exempt document; ↲
→then the person is legally entitled to access the document.
```

This is one of the trickiest parts of the new syntax, for two reasons. One reason is that the parser is not yet stable, and may not correctly parse a correctly constructed rule.

The second is that we are dealing with what is known as 'whitespace', which are characters like tabs, spaces and carriage returns which are not themselves visible, but which affect formatting. This means that it is very easy to make changes which are invisible to the rule author, but which make a big difference to the computer.

We are planning to put out updates which will make the parser more robust and fault tolerant.

6.6. Linking Rules – Metarules

Once a rule has been created for each sub-section and part of an Act the rules need to link or chain together. Rules should be linked together as they are usually only of use when inferenced collectively.

There is no strict method for linking rules via the use of metarules. Generally, a metarule will be created to fill in the gaps of connection between sections. For example, where a subsection of an Act has been split into a number of rules, each representing a part of that subsection, a metarule will be required to link the sub-sections. The following is an example of how such a metarule could look.

```
If the company is not subject to Section 21-1-a
and the company is not subject to Section 21-1-b
and the company is not subject to Section 21-1-c
then the company is compliant with Section 21-1.
```

This being the case, one of the rules could be as follows.

```
If the company does not break the law
and the company does give to charity
then the company is not subject to Section 21-1-a.
```

Make the metarule elements self explanatory. In the example above, the metarule elements were Section 21-1, Section 21-1-a, and so on. It was quite obvious that they were referring to section 21 and its sub-sections.

7. Introduction to Structured Grammar

Parsing has been previously introduced. Parsing is the splitting of clauses into structural elements. Each clause must be ‘parsed’ correctly before it is compiled into a legitimate rule. In order for a rule to be parsed, it must adhere to aide’s grammatical structure. Before designing a rulebase a budding aide developer should have a strong grasp of English grammar and an understanding of the aide grammatical structure.

When the rules are being developed from your personal knowledge, selecting the appropriate wording is not very difficult. When the rules are based upon explicit industry knowledge, selecting the appropriate wording can be far more difficult. For instance, aide is designed primarily to assist the development of knowledgebases that model legislation. These knowledgebase rules should be as similar as possible to the original legislation, while still abiding by aide's rule syntax and grammatical syntax. It is often very difficult to dissect a legislative clause into separate grammatical elements, as required by aide. Having a strong understanding of English grammar will be of significant benefit to anybody using aide to develop a legal knowledge base.

7.1. aide's Grammatical Syntax

aide follows a strict grammar structure. Each clause must comply with the grammar requirements. The grammar requirements of the current release are specified in ABNF format in Appendix B.

In general, a clause is made up of a subject (a nounphrase) and a predicate (a verb phrase).

Each condition is a single clause. For example "the book has been published". As you have seen in the Rule Syntax, rules are made up of a number of conditions or clauses, which are chained together with by logical operators.

The clauses in the FOI rules are:

```
a document is a document of an agency
the document is not an exempt document
the document is an official document of a Minister
the document is not an exempt document
the document is available for access
```

A clause begins with a noun phrase and has a verb. A clause may have an object. In the clause:

```
the document is an official document of a Minister
```

there are two noun phases:

```
the document
an official document of a Minister
```

and one verb:

```
is
```

You have been introduced to objects and their relationship to rulebase development. The aide system internally works with objects and attributes. An object is an entity like 'a document' or 'a minister'. It does not necessarily have to be physical. Each object can have multiple attributes. These are characteristics of an object like 'the document is 100 pages' or 'the minister is running late', where '100 pages' and 'running late' are attributes of the document and minister, respectively.

Every-time a noun phrase appears (for example, every-time the noun phrase 'the document' appears in the rule above), it will refer to the same object (i.e. it will refer to a particular document). aide does not care about what the article is, so all of these noun phrases will be considered the same:

```
a document
the document
that document
```

however, adjectives matter, so that all of these nounphrases will be treated differently:

```
a document
an exempt document
the document of a Minister
```

aide has quite a strict grammar. Although this grammar is designed to be close to standard formal English, it will have differences (especially at this early stage of the system development).

7.2. The Important Facts

The important facts of this are:

- Clauses are made up of a noun phrase, a verb phrase, and one or more clause objects.
- A noun-phrase may either be a simple noun-phrase such as 'the document' or 'the official document' or a complex noun phrase; for example, a noun phrase may be made up of two simple noun phrases connected by 'of', i.e. 'the official document of the Minister'.
- A verb phrase may have a simple or compound verb, an optional adverb and possibly the word 'not’.
- Clause objects may be a simple adjective ('grey'), a noun phrase ('the document') or a preposition followed by a phrase ('for access'). You may have up to three clause object phrases in a clause.

HINT

To become more familiar with English grammar look at:

http://www.grammarbook.com

http://www.edunet.com/english/grammar/index.cfm

http://www.dailygrammar.com/

http://www.gabiscott.com/bigdog/

http://englishplus.com/grammar/

http://depts.gallaudet.edu/Englishworks/grammar/

8. Working with Rulebases

In aide, a rulebase is a collection of rules which are stored in one file, and which can backchain to each other in an inferencing session. Rulebases can be loaded from webpages, or stored in files.

8.1 Compiling Rules

Once a rule has been written according to aide's grammatical rules and structural rules, you must select *compile*, so that the rule can be parsed into a format that allows the inferencing engine to understand the rule. Often a rule will adhere to aide's grammatical and structural rule, but the inferencing engine still does not immediately understand it. Such rules are said to be ambiguous. Ambiguity means that there is uncertainty or doubt as to the meaning of a thing. Therefore the one clause could be interpreted into multiple meanings.

The *aide* natural language parser is quite flexible by design. Thus, the idea is that most clauses will be able to be parsed. However, a large number of ambiguities will occur, where there are multiple ways to correctly parse the clause under the *aide* grammar.

Rather then attempting to have the parser guess which of the parses is the desired one (with all the problems inherent in using heuristics in this fashion), *aide* aims to move the problem to the user. To do this, it compares the various valid parse trees, and asks the user to resolve the differences between them.

From a user interface perspective, this means that the rule developer will be asked a series of questions about alternate parses, and will have to choose between them. *aide* only asks those questions essential to finding the correct parse.

8.2 Saving a Rulebase

Once you have created a number of rules that you wish to save as a collection, you must select *rulebase* then *save rulebase to Website.* You will be then prompted to enter a filename, usename and other details.

Aide saves the rulebases as a HTML page, and ftps the page to your designated website.

9. Error Checking

9.1. Parsing a Clause

When creating or modifying a rule you need to know whether the grammatical structure is correct. To test whether a clause is correctly structured you may highlight the clause then click and hold the right hand mouse button. This forces a menu to pop up. Keep holding the right hand mouse button until the preferred option is highlighted, then release the mouse button to activate the option.

The system will indicate whether the grammar that you have selected parses correctly, cannot be parsed, or whether it is ambiguous. If a grammatical structure is ambiguous, then it will parse but you will have to indicate which parse it the correct one when you compile the rule.

If a clause will not parse you can often use a hyphen to connect words. The current version of aide is flexible enough to allow a wide range of clause structures, however you will probably often find certain structures that cannot be parsed. For example, the following clause requires hyphens to assist parsing.

```
the organisation does not disclose the personal information
to a recipient who might-possibly-further-disclose the
personal information
```

Make sure that the words that are connected do make sense being forced together. Do not, for example, have a clause like the following.

```
The organisation-does not-disclose the-personal-information
to a recipient
```

Such a clause will not parse because there is no identifiable noun and the parser would expect that “The organisation-does” is the subject.

Appendix A: Shortcut Keys (Windows Version)

Shortcut	Result
Alt - X	Cuts highlighted text from screen into clipboard.
Alt – V	Pastes the contents of clipboard at the position of the cursor.
Alt – C	Copies highlighted text into clipboard and leaves the text highlighted on the screen.
Alt – A	Selects all the text in the window.
Alt – U	Changes the justification between left, right and center.
Ctrl – C	Compiles the current rule.
Ctrl – W	Select all the text in the current line.
Ctrl – R	Indent the line.
Ctrl – K	Change font.

Appendix B: *aide* Grammar

The modified ABNF (rfc2234) used is defined as:

[...]	apply zero or one times;
... / ...	choose one of the alternatives;
"..."	use the literal characters enclosed;
(...)	used for grouping.

```
Clause                  =   Subject Predicate
Subject                 =   NounPhrase / "there"
Predicate               =   VerbPhrase

NounPhrase              =   [Article] [Adjective] [Adjective] noun
                            [AdjectivalModifier] [AdjectivalModifier]
VerbPhrase              =   (SimpleVerb / CompoundVerb) [Object]
                            [AdverbialModifier] [AdverbialModifier]

Object                  =   NounPhrase

CompoundVerb            =   AuxiliaryVerb ["not"] [AuxiliaryVerb] [adverb]
                            verb
SimpleVerb              =   [adverb] verb ["not"]

AdjectivalModifier      =   AdjectivalPhrase / AdjectivalClause
AdjectivalPhrase        =   PrepositionalPhrase / ParticiplePhrase /
                            (Preposition Pronoun Adjective)
AdjectivalClause        =   ("that" Clause) / (Preposition Pronoun Clause)

AdverbialModifier       =   AdverbialPhrase / AdverbialClause
AdverbialPhrase         =   PrepositionalPhrase / ParticiplePhrase
AdverbialClause         =   "to" VerbPhrase

PrepositionalPhrase     =   Preposition NounPhrase
ParticiplePhrase        =   Participle NounPhrase
```

Parts of Speech:

The following parts of speech are defined:

```
Article             =   "a" / "an" / "the" / "these" / "this" / "some" / "any"

Pronoun             =   "that" / "who" / "whom" / "which"

Preposition         =   "as" / "about" / "above" / "across" / "after" /
                        "against" / "along" / "among" / "around" / "at" /
                        "before" / "behind" / "below" / "beneath" / "beside" /
                        "between" / "beyond" / "by" / "despite" / "down" /
                        "during" / "except" / "for" / "from" / "in" / "inside" /
```

```
                    "into" / "like" / "near" / "of" / "off" / "on" / "onto"
                    / "out" / "outside" / "over" / "past" / "since" /
                    "through" / "throughout" / "till" / "to" / "toward" /
                    "under" / "underneath" / "until" / "up" / "upon" /
                    "with" / "within" / "without" / "while"

AuxiliaryVerb     =  "can" / "could" / "may" / "might" / "must" / "ought" /
                    "should" / "would" / "will" / "do" / "does" / "did" /
                    "done" / "been" / "am" / "is" / "are" / "was" / "were" /
                    "have" / "had" / "has" / "be" / "used"
```

Other parts of speech are as defined in the Mitton Corpus (Computer Readable version of the Oxford Learner's Dictionary). The corpus can be found in the Oxford Text Archives.

Appendix C: *aide* Architecture